Programmation Multi-Tâches

Document de travail

Dimitry SOLET

19 avril 2023

ESEO Apprentissage

Plan

Présentation du module

Objectifs :

- Comprendre les enjeux liés à la programmation multitâche :
  - Pourquoi en faire ? Quelles sont les difficultés ?
- Savoir identifier les situations de concurrence.
- Proposer des constructions permettant de pallier aux situations de concurrence.
- Mettre en œuvre les concepts du multitâche sous Linux.

Modalités pédagogiques :

- Entrelacement de cours magistraux et d'activités pratiques.
  - Cours magistraux : présentation des concepts théoriques.
  - Activités pratiques : mise en application des concepts selon le standard `POSIX`.
- Évaluation lors de la dernière séance.

Fonctionnement des séances

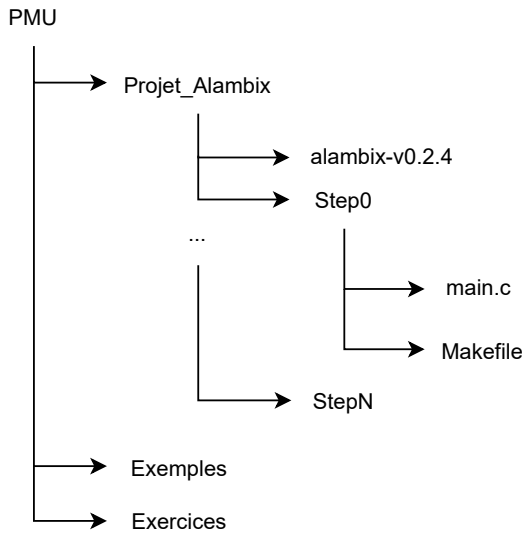
Les activités pratiques :

- *Lecture* : Cours sur la programmation multitâche en langage C selon le standard `POSIX`.
- *Exploration* : Ensemble d'exemples et d'exercices permettant de s'approprier les différentes notions.
- *Alambix* : Application ≪ fil rouge ≫ permettant de mettre en œuvre les différentes notions.

Utilisation d'un code couleur :

Lecture **Exploration** **Alambix**

Organisation du répertoire :



Planning prévisionnel

Séance 1	Introduction au module Cours magistral : Généralités Activités pratiques : Processus
Séance 2	Activités pratiques : Processus Activités pratiques : Les threads
Séance 3	Cours magistral : La concurrence Activités pratiques : Mutex et barrière
Séance 4	Activités pratiques : Mutex et barrière Cours magistral : Communication Activités pratiques : Les boîtes aux lettres
Séance 5	Activités pratiques : Les boîtes aux lettres
Séance 6	Activités pratiques : Les signaux Cours magistral : Watchdog Activités pratiques : Temporisation et watchdog
Séance 7	Activités pratiques : Temporisation et watchdog Activités pratiques : Fin d'Alambix
Séance 8	Activités pratiques : Exercices complémentaires Activités pratiques : Évaluation

Début avec Alambix

Objectif pédagogique

Apprendre par l'expérimentation ~~la gestion d'un bar et~~ la programmation multitâche sous un système Unix.

Les dépendances

- gcc, make
- pkg-config, libgtk-3-dev, libx11-dev

Alambix

→ Télécharger et placer la lib d'alambix dans votre dossier de travail.

→ Exécuter la commande

```
firefox alambix-v0.2.4/doc/html/index.html &
```

- Appliquer Step 0.

Plan

Les processus 1/4

Cours : Lire les diapositives 1 à 6 (système)

Exploration

Vérifier que le `PID` d'Alambix est le même en console et dans l'IHM.

Cours : Lire la diapositive 6 (programmation)

Exploration

- → Analyser et exécuter `ex_01_fork.c`
- → Écrire un programme (`exo_1.c`) permettant d'exécuter 5 processus fils. Chacun d'entre eux devra afficher 5 fois d'affilé son numéro d'ordre entre 0 et 4.

Les processus 2/4

Cours : Lire la diapositive 7

Exploration

Écrire un programme (`exo_2.c`) permettant de lancer le programme `/usr/bin/baobab` avec `execl`, puis `execv` et `execvp`.

→ Que se passe-t-il au niveau de la console ?

→ Dans la console, faire `CTRL+C`. Qu'observe-t-on ?

Cours : Lire la diapositive 8

Exploration

→ Analyser et exécuter `ex_02_fork_exec.c`

→ Observer le lien de filiation (`ps axj`)

Les processus 3/4

Alambix

Appliquer Step 1.

Cours : Lire la diapositive 9

Exploration

→ Lire la documentation de la fonction `exit()` : `man 3 exit`

→ Analyser et exécuter `ex_04_fork_exec_exit.c`

- Observer le processus fils quand le père se termine : `ps j <id_fils>`

Cours : Lire la diapositive 10

Les processus 4/4

Exploration

Toujours avec `ex_04_fork_exec_exit.c`

- Tuer (commande `kill`) le processus fils puis observer son état avant 15 secondes.
- Observer le processus fils après 15 secondes.

Analyser et exécuter `ex_05_fork_exec_wait.c`

- Que peut-on constater sur le moment de terminaison du processus père ?

Exploration

Écrire un programme qui lance dix fils qui effectuent une « course » et qui affiche à la fin l'ordre des fils (pid et ordre d'activation). Chaque fils effectuera *n* tours d'une boucle vide. *n* est choisit au hasard entre 5000 et 10000 (`exo_3.c`).

Plan

Les threads 1/4

Cours : Lire les diapositives 12 à 13

Exploration

Analyser et exécuter `ex_06_pthread_create.c`

- Observer les threads avec la commande `ps maux`.
- Modifier l'exemple de façon à ce que le thread principal se termine avant les autres threads.

Cours : Lire la diapositive 14

Exploration

Analyser et exécuter `ex_07_pthread_exit.c` pour les deux versions (voir commentaires dans le code)

- Quelle différence peut-on observer entre les deux versions ?

Les threads 2/4

Alambix

Relever le défi niveau 1 (version 1).

→ Appuyer sur `play again`

→ Quels problèmes peut-on constater ?

Cours : Lire la diapositive 15

Exploration

- Analyser et exécuter `ex_08_pthread_join.c`

Les threads 3/4

Exploration

- Écrire un programme (`exo_4.c`) qui créé un thread pour calculer la moyenne d'un tableau d'entiers. Ce thread doit calculer la moyenne et le `main` s'occupera d'afficher le résultat.
  - Aucun argument n'est passé au thread.
  - Le thread ne retourne aucune valeur.
  - Il est autorisée d'utiliser des variables globales.
- Modifier le code afin de ne pas avoir de variable globale. Il est maintenant possible de passer un argument au thread et de récupérer sa valeur de retour.

Les threads 4/4

Exploration

- Analyser et exécuter `ex_09_pthread_detach.c`

Alambix

Relever le défi niveau 1 (version 2).

→ Corriger le problème de mémoire.

Cours : Lire les diapositives 16 à 17

Exploration

Analyser et exécuter `ex_10_pthread_attr.c`

Plan

Synchronisation : Mutex

Cours : Lire les diapositives 18 à 28

Exploration

(`exo_5.c`) Écrire un programme qui crée deux threads. Chaque thread doit incrémenter un compteur partagé 100000 fois. À la fin de l'exécution des threads, le programme doit afficher la valeur du compteur partagé.

- Mettre en évidence le problème observé.

Utiliser un mutex afin de corriger le problème.

Alambix

Terminer le défi niveau 1 (version 3).

Synchronisation : Barrière de synchronisation

Alambix

Relever le défi niveau 2 (version 1).

→ Quel est le problème ?

Cours : Lire les diapositives 34 à 35

Exploration

Analyser et exécuter `ex_12_pthread_barrier.c`

Alambix

Relever le défi niveau 2 (version 2) avec une barrière.

→ Corriger le problème avec l'utilisation d'une barrière

Synchronisation : Sémaphore

Cours : Lire les diapositives 36 à 41

Exploration

Analyser et exécuter `ex_13_sem.c`

- Lancer l'exécutable dans deux shells
- Observer le contenu du dossier `/dev/shm`

Alambix

Relever le défi niveau 2 (version 3) avec des sémaphores anonymes.

Exploration

Développer une barrière de synchronisation avec un sémaphore et un mutex. (`exo_6.c`)

Communication : Files de messages

Cours : Lire les diapositives 48 à 54

Exploration : Analyser et exécuter `ex_15_mq.c`

Alambix : Relever le défi niveau 3

- version 1 : Initialiser une file de message et gérer l'envoie des message.
  - Observer ce qui se passe en redémarrant l'application.
  - Corriger le problème manuellement.
- version 2 : Corriger le problème observé.
  - Interrompre le programme (`Ctrl+C`) après avoir déposé le 1er message, tester le redémarrage de l'application.
- version 3 : Corriger le problème observé.

Alambix : Relever le défi niveau 4

.

Communication : Les signaux 1/2

Cours : Lire les diapositives 58 à 59

Exploration : Analyser et exécuter `ex_16_signal_list.c`

Cours : Lire les diapositives 60 à 61

Exploration : Analyser et exécuter `ex_17_fork_exec_wait_signal.c`

- Tester `Ctrl+C`.

Cours : Lire les diapositives 62 à 66

Exploration : Analyser et exécuter `ex_18_fork_exec_wait_sigaction.c`

- Tester `Ctrl+C`

Communication : Les signaux 2/2

Alambix

Ajouter un gestionnaire de signal pour éviter l'apparition de zombis suite à l'ouverture de l'aide (avec `sigaction`).

Exploration

- `exo_10.c` Écrire un programme qui crée un fils qui fait un calcul sans fin. Le processus père propose alors un menu :
  - L'appuie sur la touche $'s'$ endort le fils.
  - L'appuie sur la touche $'r'$ redémarre le fils.
  - L'appuie sur la touche $'q'$ tue le fils puis termine le père.

BONUS : Communication : Les tubes

Cours : Lire les diapositives 55 à 57

Exploration : `exo_7.c`, `exo_8.c`, `exo_9.c`

- Écrire un programme qui créé un tube, puis écrit une donnée de type `char*` dedans et enfin lit cette donnée une par une.
- Écrire un programme composé d'un père et d'un fils. Le père récupère une chaîne de 20 caractères au clavier puis l'envoie dans le tube. Le fils récupère la chaîne de caractère puis l'affiche à l'écran.
  - Il faut fermer la partie du tube qui n'est pas utilisée avec `close()`.
- Écrire un programme créant trois processus : deux écrivains et un lecteur. Les deux écrivains écrivent respectivement les séquences *ABC...Z* et *abc...z* par bloc de trois caractères (attendre 3 secondes entre les blocs). Le lecteur lit dans le tube par bloc de 4 caractères et affiche ce qu'il lit.

Plan

Temporisation

Fin des diapositives

Exploration : Analyser et exécuter `ex_19_timer.c`

Alambix : Relever le défi niveau 5

- Utiliser un minuteur (`timer_t`).

Exploration : Développement d'un *Watchdog*

- Compléter le code du module `watchdog` et vérifier son fonctionnement.
- `exo_11.c` Faire évoluer le programme d'utilisation du `watchdog` afin que :
  - Le calcul de la boucle main soit arrêté par le *watchdog*.
  - L'appuie sur `Ctrl+C` annule le *watchdog*.

Fin d'Alambix

Alambix : Relever les défis niveau 6 et 7

Plan

Select

La fonction `select`

La fonction `select( )` permet de surveiller plusieurs descripteurs de fichier, et indique lequel est prêt à être lu (ou écrit).

Lecture

Lire la partie : *Création d'un pipe dans processus unique* du lien

Activité

Faire l'activité *Attente multiple avec « select »* (disponible sur le campus).